Dog Breed classifier Udacity Capstone Project

Project Overview

This project is all about the classic Machine Learning classification. Given a dataset of random images, our model should be able to detect the breed of the dog if a dog is present in the image. We are going to use **CNN (Convolutional Neural Network)** to build this model.

Initially we are going to import some of the necessary modules required for this project. Then we are going to download the required datasets containing images required for this project. Firstly we are going to try and detect human and dog faces in the datasets before proceeding to the project to get an initial idea. The data is then pre-processed by horizontally flipping, cropping the images and so on and so forth. Python Libraries like TensorFlow and openCV to detect the faces in the images.

We are going to use CNN to classify dog breeds (using Transfer Learning), then write, train and test the algorithm.

Problem Statement

When a random image is fed into the model, it will classify the image to detect the dog and it's breed in the image and return to the user.

Importing Modules and datasets

Link to the datasets https://github.com/Gyeah3/dog-breed-classifier-udacity/tree/master/project-dog-classification/images

Pytorch will be the main machine learning (ML) library used for this project. The benchmark model for this project will be the **VGG-16**, a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford, this model achieved 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. Achieving 92.7% accuracy for this project with the small dataset available (8351 images of dogs) in comparison to the image net data set which the VGG-16 was trained is definitely a far reach, but it’s something to aim for. First we are going to import all the necessary modules and libraries required for the project.

We are going to detect the human faces and dog faces in the image datasets by converting the images into numpy arrays.

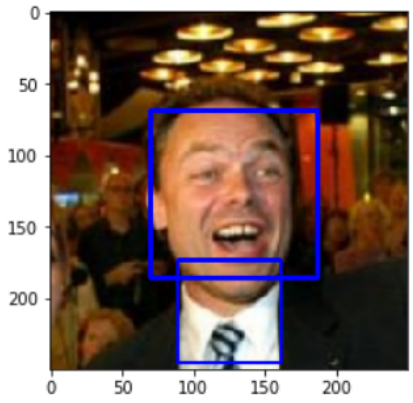
```
import numpy as np
from glob import glob

# load filenames for human and dog images
human_files = np.array(glob("lfw/*/*"))
dog_files = np.array(glob("dogImages/*/*/*"))

# print number of images in each dataset
print('There are %d total human images.' % len(human_files))
print('There are %d total dog images.' % len(dog_files))
```

```
There are 13233 total human images.
There are 8351 total dog images.
```

We are going to use openCV's implementation to detect the faces in the datasets. openCV provides many pre-trained face detectors and we are going to download one such detector in the harrcascades directory.

```
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# extract pre-trained face detector
face_cascade = cv2.CascadeClassifier('haarcascades/haarcascade_frontalface_alt.xml

# load color (BGR) image
img = cv2.imread(human_files[0])
# convert BGR image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# find faces in image
faces = face_cascade.detectMultiScale(gray)

# print number of faces detected in the image
print('Number of faces detected:', len(faces))

# get bounding box for each detected face
for (x,y,w,h) in faces:
    # add bounding box to color image
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

# convert BGR image to RGB for plotting
cv_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# display the image, along with bounding box
plt.imshow(cv_rgb)
plt.show()
```

```
Number of faces detected: 2
```



Then we write a function which returns **true** on detecting human face.

```
# returns "True" if face is detected in image stored at img_path
def face_detector(img_path):
    img = cv2.imread(img_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)
    return len(faces) > 0
```

Detect Dogs

To detect the dogs, we are going to use a pre-trained model, in this case VGG-16 model. The categories corresponding to dogs in the VGG-16 model appear in an uninterrupted sequence and correspond to index 151–268, inclusive. Thus, in order to check to see if an image is predicted to contain a dog by the pre-trained VGG-16 model, we need only check if the pre-trained model predicts an index between 151 and 268 (inclusive).

Images needs to be pre-processed to tensors before being passed to a neural network, the process_image function below handles this task.

```
from PIL import Image
import torchvision.transforms as transforms

def load_image(img_path):
    image = Image.open(img_path).convert('RGB')
    # resize to (244, 244) because VGG16 accept this shape
    in_transform = transforms.Compose([
                        transforms.Resize(size=(244, 244)),
                        transforms.ToTensor()]) # normalizaiton parameters from pytorch doc.

    # discard the transparent, alpha channel (that's the :3) and add the batch dimension
    image = in_transform(image)[:3,:,:].unsqueeze(0)
    return image
```

```
from PIL import Image
import torchvision.transforms as transforms

# Set PIL to be tolerant of image files that are truncated.
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

def VGG16_predict(img_path):
    '''
    Use pre-trained VGG-16 model to obtain index corresponding to
    predicted ImageNet class for image at specified path

    Args:
        img_path: path to an image

    Returns:
        Index corresponding to VGG-16 model's prediction
    '''

    ## TODO: Complete the function.
    ## Load and pre-process an image from the given img_path
    ## Return the *index* of the predicted class for that image
    img = load_image(img_path)
    if use_cuda:
        img = img.cuda()
    ret = VGG16(img)
    return torch.max(ret,1)[1].item() # predicted class index
```

```
# predict dog using ImageNet class
VGG16_predict(dog_files_short[0])
```

```
181
```

Creating a CNN to classify the dog breeds

Now that we have functions for detecting humans and dogs in images, we need a way to predict breed from images. In this step, you will create a CNN that classifies dog

breeds. We must create our CNN *from scratch* (so, you can't use transfer learning *yet*!), and we must attain a test accuracy of at least 1%.

Use a CNN to Classify Dog Breeds (using Transfer Learning)

To reduce training time without sacrificing accuracy, we show you how to train a CNN using transfer learning. In the following step, you will get a chance to use transfer learning to train your own CNN.

Bottleneck features are implemented on pre-trained CNN. I picked ResNet as a transfer model because it performed outstanding on Image Classification. I looked into the structure and functions of ResNet. The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more layers. I guess this prevents overfitting when it's trainin

```
for param in model_transfer.parameters():
    param.requires_grad = False

model_transfer.fc = nn.Linear(2048, 133, bias=True)


fc_parameters = model_transfer.fc.parameters()


for param in fc_parameters:
    param.requires_grad = True


model_transfer
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
      (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (downsample): Sequential(
        (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
```

Train and test the model

Since this is a classification problem with no catastrophic consequences for getting a prediction wrong, we will be using accuracy as our evaluation metrics. It is the ratio of number of correct predictions to the total number of input samples. Accuracy an evaluation metric works well for a balanced dataset which happens to be the case for the dataset for this project. We train and test the model. DUring the training we run 10 epochs and the save the model with the lowest validation loss for a better accuracy.

```
# train the model
# train(n_epochs, loaders_transfer, model_transfer, optimizer_transfer, criterion_transfer, use_cuda, 'model_transfer.pt')
def train(n_epochs, loaders, model, optimizer, criterion, use_cuda, save_path):
    """returns trained model"""
    # initialize tracker for minimum validation loss
    valid_loss_min = np.Inf

    for epoch in range(1, n_epochs+1):
        # initialize variables to monitor training and validation loss
        train_loss = 0.0
        valid_loss = 0.0

        ##################
        # train the model #
        ##################
        model.train()
        for batch_idx, (data, target) in enumerate(loaders['train']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()

            # initialize weights to zero
            optimizer.zero_grad()

            output = model(data)

            # calculate loss
            loss = criterion(output, target)

            # back prop
            loss.backward()

            # grad
            optimizer.step()

            train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss))

            if batch_idx % 100 == 0:
                print('Epoch %d, Batch %d loss: %.6f' %
                  (epoch, batch_idx + 1, train_loss))

        ######################
        # validate the model #
        ######################
        model.eval()
        for batch_idx, (data, target) in enumerate(loaders['valid']):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            ## update the average validation loss
            output = model(data)
            loss = criterion(output, target)
            valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss))


        # print training/validation statistics
        print('Epoch: {} \tTraining Loss: {:.6f} \tValidation Loss: {:.6f}'.format(
            epoch,
            train_loss,
            valid_loss
            ))

        ## TODO: save the model if validation loss has decreased
        if valid_loss < valid_loss_min:
            torch.save(model.state_dict(), save_path)
            print('Validation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.format(
            valid_loss_min,
            valid_loss))
            valid_loss_min = valid_loss

    # return trained model
    return model
```

```
train(20, loaders_transfer, model_transfer, optimizer_transfer, criterion_transfer, use_cuda, 'model_transfer.pt')
```

```
Epoch 1, Batch 1 loss: 3.306457
Epoch 1, Batch 101 loss: 3.309227
Epoch 1, Batch 201 loss: 3.311080
Epoch 1, Batch 301 loss: 3.289307
```

Writing our algorithm

Write an algorithm that accepts a file path to an image and first determines whether the image contains a human, dog, or neither. Then,

```
if a dog is detected in the image, return the predicted breed.
if a human is detected in the image, return the resembling dog breed.
if neither is detected in the image, provide an output that indicates an error.
```

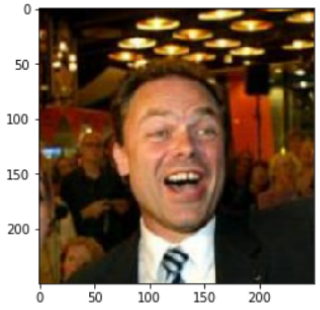
```
def run_app(img_path):
    ## handle cases for a human face, dog, and neither
    img = Image.open(img_path)
    plt.imshow(img)
    plt.show()
    if dog_detector(img_path) is True:
        prediction = predict_breed_transfer(model_transfer, class_names, img_path)
        print("Dogs Detected!\nIt looks like a {0}".format(prediction))
    elif face_detector(img_path) > 0:
        prediction = predict_breed_transfer(model_transfer, class_names, img_path)
        print("Hello, human!\nIf you were a dog..You may look like a {0}".format(prediction))
    else:
        print("Error! Can't detect anything..")
```

```
for img_file in os.listdir('./images'):
    img_path = os.path.join('./images', img_file)
    run_app(img_path)
```

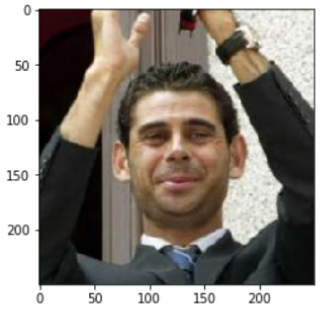
Testing the algorithm

Now that we have trained our CNN model and written an algorithm to specify the file path to an image to determine what the image contains, now it's time to test the algorithm and see some interesting results.

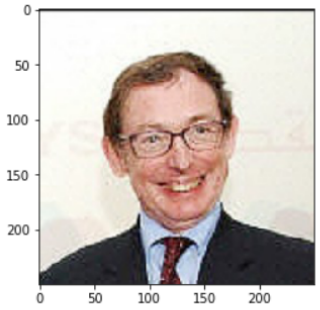
```
my_human_files = ['./my_images/human_1.jpg', './my_images/human_2.jpg', './my_images/human_3.jpg' ]
my_dog_files = ['./my_images/dog_shiba.jpeg', './my_images/dog_yorkshire.jpg', './my_images/dog_retreiver.j

for file in np.hstack((human_files[:3], dog_files[:3])):
    run_app(file)
```



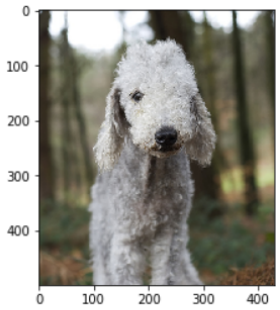
```
Hello, human!
If you were a dog..You may look like a Chinese crested
```



```
Hello, human!
If you were a dog..You may look like a American staffordshire terrier
```



```
Hello, human!
If you were a dog..You may look like a Irish wolfhound
```



```
Dogs Detected!
It looks like a Bedlington terrier
```

Conclusion and possible improvements

I started out trying to create a CNN with more than 60% testing accuracy under 2 mins of training on GPU. Our final model obtained more than 70% testing accuracy under the training of 20 secs.

There are still very high chances to increase model accuracy with the following techniques:

Image Augmentation, Increasing Dense layers and Increasing no of epochs with Dropout to decrease the chances of model overfitting.

Following the above areas, I’m sure we could increase the testing accuracy of the model to above 95%.

Overall, this project and Udacity's nano-degree has been an eye-opener on how deep machine learning can go and how much more I need to learn. I have absolutely had so much fun doing this project.